Predator-Prey Game

Group # 48

Katrine Chow : Adam Benckeser : Daniel Church : Vinny Harris-Riviello

Design Plan

A super duper amazing plan ...

Bug World class

Has a grid that is a dynamic array of pointers to Critter, and a Menu object. It represents the world were the bugs interact with each other and follow their dynamic to survival, moving through the grid.

BugWord class

Member variables: Critter *** grid[]; Menu menu; timeSteps
Member functions: Constructors printGrid() sortGrid() isEmpty() getEmptyLocation()

Critter class (base class)

Critter class is the base class for ant and doodlebug. The two classes will inherit variables and functions that they share from their base class. The ant and doodlebug are derived

classes.

Member variable (can be id or symbol will be of char type, which will be for ant and doodlebug can either ‘X’ or ‘O’).

The variable timeSteps hold the number of steps/rounds it needs to stay alive to be able to breed. Each critter has its own specific cycle. So, they would have to be defined in the derived class. Current hold the timeSteps that have passed so far, it initializes to 0.

Critter class

Member variables: id/symbol/name? timeSteps = 0;
Member functions: Constructors: Critter(char, int). Getters and setters for members variables. virtual move(Critter *, int) breed()

move() - part of the functionality of move can be inherited to the derived classes even if their version gets called, but calling the parent move() since there is behavior that is similar for both ant and doodlebug. Move function will have a parameter list, with a pointer to an array (grid) and an int for size of the array.

breed() - also have same functionality in the derived with a few differences, but can create template in base class.

Ant Class (derived class)

The ant class has a timeSteps member variables to hold the number of steps that it needs to stay alive to breed.

Ant class

Member variables

Inherited: id/symbol/name? Int timeSteps = 0;

Member functions:
Constructors
move() override;
breed();

Ant() - constructor needs to initialize the base class, passing 3 to the the timeSteps in an explicit way. And also it's assign symbol.

Doodlebugs class (derived class)

The Doodlebug class has a timeSteps member variables to hold the number of steps that it needs to stay alive to breed, just like Ant inheriting this from Critter.

Doodlebug class

Member variables inherited: id/symbol/name? Int timeSteps = 0;

Member functions:
Constructors
move() override;
breed();
starve();

Doodlebug() - constructor needs to initialize the base class, passing 8 to the the timeSteps in an explicit way. And also it's assign symbol.

starve() doodlebugs an only go 3 steps/rounds without eaten an ant otherwise they starve and die. The member variable will be called to check how many timeSteps have passed and determine the fate of the bug from this.

Menu Class

Menu gets settings from the user and lets game know when to start, or if the user chooses to quit, it ends the game. Menu prompt the user to enter the number of time steps to run. Menu is also design for the extra credit, so it also prompts the user to enter the size of the grid rows and columns, the number of ants, and the number of doodlebugs. The Menu class handles all the inputs that are required to run the game. An instance of a menu is initialized in the constructor of bugWorld.

Menu

Member variables inherited:
Member functions: welcome() userRows() userCols() userAnts() userDBugs() continueGame() validateInteger(int, int) userTimeSteps()

The various functions provide the info needed for the extra credit and the time steps for the game. Welcome is the introduction to the game and specifies the extra credit is included, continueGame allows the user to exit after their specified number of time steps or enter a new number, and validateInteger validates the inputs.

When the game begins all values of the struct UserSettings should be already initialized. The game will use this settings to configure the bug world.

Testing

Description	Input	Expected Result
User starting inputs	Rows, Cols, Ants, Bugs, steps	Input validated and passed to bugWorld.
Critters random generator	N/A. Ants and Doodlebugs generated at random locations on the grid	Grid is populated by critters in randomized locations.
Ant move() moves all Ants on the board to adjacent empty spaces	N/A function call to moveBugs().	Ants are moving as expected.
Doodlebug move() moves all Doodlebugs on the board to adjacent empty spaces. Eats if space occupied by Ant	N/A function call to moveBugs().	Doodlebugs are moving as expected.
Doodlebug starve(). If doodlebug has not eaten in 3 timesteps, starves to death.	N/A function call to starve()	Doodlebugs die if no food for 3 timesteps.
Ant breed() - if Ant survives 3 timesteps, breeds a new Ant at empty adjacent space. If no empty spaces, no breeding occurs	N/A function call to breed()	Ants breed after 3 timesteps and when there is an empty adjacent cell.
Doodlebug breed() - If Doodlebug survives for 8 timesteps, breed a new Doodlebug at empty adjacent space. If no empty spaces, no breeding occurs.	N/A function call to breed()	Doodlebugs breed after 8 timesteps and when there is an empty adjacent cell.
Ants stay within grid boundaries	N/A Ant::move()	Ants are not passing beyond grid.
Doodlebugs stay within grid boundaries	N/A Doodlebug::move()	Doodlebugs are not passing beyond grid.
Continue/Exit	1 to Continue, 2 to Exit	1 continues game; 2 exits

Reflection

Working on a bigger project becomes way more complex with a team. We needed to organize which software to use, how to divide the work, what type of design to implement, and make sure that we were covering all the requirements.

We chose to use Cloud 9 mainly because it seemed like a good platform to collaborate online. We voted over a private channel that we opened in Slack and we used this to communicate throughout the project completion. Cloud 9 also was conducive to a type of collaboration that involved our own definition of pair programming. We separated in two pairs. And one pair worked on the menu and bug world classes and the other pair in the critter, ant, and doodlebug classes.

Since we are using polymorphism to handle the ants and doodlebugs. It made for a leaner code than if we had to constantly repeat functions and configurations for each. It was challenging however to place the bugs in the grid in its original state, since we needed to check that there we not repeated placements.

Working on code as a team was very challenging. Getting on the same page of how we wanted to flow in the program, took time and effort. It definitely was a learning experience.

One of the major changes that we encountered, is that in order to have polymorphism in a 2D array, the grid, we needed a pointer that would point to an array of pointers. So, we had to modify the design to be Critter***. This happened when we had already started the project and we ran into the situation of assigning the ant pointer for example to the Critter pointer. then you would find a Critter not a pointer to a Critter. The necessary condition to actually have polymorphism is that you switch pointers, not actual values of the derived class and the base class. So, we had to accommodate the program to this change.

Additionally, as each team member had different schedules and responsibilities outside of school, it became extremely difficult to get together for regular updates. There were also several parts of the code that we were unable to test until other portions are completed. These challenges encouraged us to improve delegation of

tasks and project planning.

Overall this has been a valuable learning experience that highlighted the differences between working individually and in a group setting. Although we had a slow start, the team pulled through in the end thanks to the members' dedication and focus. (Special shout out to Vinny, our fearless leader!).